МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ML-СЕРВИСА ДЛЯ ПРИЛОЖЕНИЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ УЧЕБНЫХ КАРТОЧЕК

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Дружкина Дениса Ивановича

Научный руководитель
зав. каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2026

Современный мир переживает этап глубокой цифровой трансформации, которая охватывает все сферы человеческой деятельности — от экономики и производства до повседневного общения и, конечно, образования. Цифровизация образования давно перестала быть просто трендом, она стала объективной реальностью. Традиционные меловые доски и бумажные конспекты уступают место интерактивным онлайн-курсам, облачным хранилищам материалов и мобильным приложениям. В этом контексте смартфон или планшет выступают не только как средство развлечения или связи, но и как полноценный инструмент для учебы, позволяющий получать знания в любом месте и в любое время.

Какими бы разными ни были современные образовательные технологии, конечная цель для большинства студентов и школьников остается классической — успешная сдача экзаменов, зачетов или контрольных работ. Объем информации, которую необходимо усвоить за ограниченный промежуток времени, часто бывает колоссальным. Простое перечитывание учебников или просмотр лекций, как показывают исследования, оказываются малоэффективными для долговременного удержания материала. Информация, полученная однократно, имеет свойство быстро забываться, если не предпринимать специальных мер для ее закрепления. Именно здесь на сцену выходят технологии искусственного интеллекта, которые способны существенно ускорить и упростить отдельные этапы учебного процесса. В частности, применение LLM (больших языковых моделей) позволяет автоматизировать наиболее трудоемкую рутинную операцию — создание учебных карточек, превращая неструктурированный текст лекций или конспектов в готовые вопросно-ответные пары и тем самым снижая порог входа в технологии интервального повторения.

В рамках работы осуществляется разработка мобильного приложения SigmaCards с ML-сервисом для автоматической генерации учебных карточек из текстов, PDF-файлов и изображений с использованием методов интервального повторения.

Целью данной работы является разработка и интеграция ML-сервиса в мобильное приложение SigmaCards. Для достижения поставленной цели были поставлены следующие задачи:

— исследовать методы и подходы к автоматической генерации учебных вопросов и ответов на основе больших языковых моделей;
— проанализировать современные open-source решения для оптического рас-

познавания текста (Vision OCR), ориентированные на кириллицу, и выбрать наиболее подходящее для интеграции;

— разработать архитектуру ML-сервиса как отдельного, масштабируемого компонента, обеспечив его асинхронное взаимодействие с основным сервером SigmaCards через брокер сообщений Apache Kafka;
— реализовать программный конвейер обработки: загрузка файла, извлечение текста с помощью OCR модели и последующая генерация карточек через LLM.
— интегрировать разработанный ML-сервис в состав приложения SigmaCards и написать документацию.

В первом разделе работы рассматривается предметная область. Психологической основой системы служит кривая забывания Эббингауза, открытая в конце 1800-х годов. Опыты показали, что уже через 20 минут после заучивания испытуемые воспроизводили лишь 60% материала, спустя девять часов — 40%, а по прошествии месяца — около 20%. Память человека подвержена экспоненциальному закону распределения: наиболее интенсивная потеря информации происходит в первые часы и дни после первоначального изучения, после чего скорость забывания постепенно замедляется.

Эббингауз не только описал проблему забывания, но и предложил путь ее решения. Его исследования показали, что своевременное повторение изученного материала способно существенно замедлить процесс забывания. Более того, если повторения осуществляются с постепенно увеличивающимися интервалами, эффективность запоминания значительно возрастает по сравнению с равномерным или «зубрежным» повторением материала за короткий промежуток времени.

Это явление получило название эффекта распределенной практики. Как отмечается в современных обзорах, эффект интервалов считается одним из наиболее надежно подтвержденных феноменов в психологии обучения. Многочисленные исследования, проведенные после работ Эббингауза, подтвердили устойчивость этого эффекта для различных типов учебного материала — от словарных пар до сложных концептуальных знаний — и для разных возрастных групп обучающихся.

Эффект распределенной практики объясняется несколькими когнитивными механизмами. Во-первых, временные интервалы между повторениями

способствуют процессам консолидации памяти — после завершения учебной сессии мозгу необходимо время для стабилизации новых нейронных связей и интеграции новой информации в существующие когнитивные структуры. Во-вторых, каждое последующее обращение к материалу требует от обучающегося активного воспроизведения информации из памяти, что является когнитивно более затратным процессом по сравнению с простым перечитыванием. Именно эта усиленная когнитивная нагрузка способствует более глубокому и прочному закреплению материала. В-третьих, распределенные повторения обеспечивают вариативность контекста: информация извлекается в разных условиях, что способствует формированию более гибких и применимых в новых ситуациях ментальных репрезентаций.

Дополняет этот подход эффект тестирования (retrieval practice effect): попытка самостоятельно извлечь информацию из памяти, например путем ответа на вопрос карточки, приводит к более прочному закреплению материала, чем его пассивное перечитывание.

В классическом эксперименте Родигера и Карпике было показано, что студенты, которые после изучения текста проходили тестирование по его содержанию, демонстрировали значительно лучшую отсроченную воспроизводимость информации по сравнению с теми, кто просто перечитывал текст несколько раз. Примечательно, что даже в тех случаях, когда ответ был неверным, сам процесс попытки извлечения информации оказывал положительное влияние на последующее запоминание — особенно если за ним следовала корректирующая обратная связь.

Далее в этом же разделе рассматриваются алгоритмы интервального повторения. Алгоритм SM-2, предложенный Петром Возняком в конце 1980-х годов, стал первым широко распространенным алгоритмом такого рода. Его суть заключается в поддержании для каждой карточки двух параметров: интервала повторения и фактора легкости. После каждого показа пользователь оценивает сложность воспроизведения по шкале от 0 до 5, и следующий интервал рассчитывается умножением текущего на фактор легкости. SM-2, однако, страдает от упрощенной модели памяти: он не различает стабильность запоминания и сложность материала. Более современной альтернативой является алгоритм FSRS (Free Spaced Repetition Scheduler, 2022), использующий трехпараметрическую DSR-модель памяти — стабильность, сложность и воспроизводимость.

FSRS явно вычисляет текущую вероятность успешного воспроизведения и превосходит SM-2 по точности прогнозирования забывания. Именно FSRS выбран в системе SigmaCards в качестве основного планировщика повторений.

В подразделе анализа существующих решений рассматриваются три ключевые платформы: Anki, Quizlet и Duolingo.

Anki является одной из старейших и наиболее уважаемых в сообществе SRS-платформ. Программа использует интервальное повторение на основе кривой забывания Эббингауза, применяя современный FSRS. Anki распространяется бесплатно на всех платформах, за исключением iOS (где приложение стоит 2290 руб.), и полностью открыт для расширений через систему аддонов. Однако именно эта открытость и модульность оборачиваются и недостатком: сценарий автоматической генерации карточек из PDF или фотографий не является штатным процессом. Пользователю приходится самостоятельно подбирать и настраивать сторонние дополнения для OCR или импорта, что требует технической подготовки. Более того, поддержка распознавания кириллического текста реализована лишь в отдельных аддонах и не гарантирует стабильного качества.

Quizlet представляет собой современную облачную платформу, которая в последнее время активно внедряет AI-инструменты для генерации учебных материалов. Пользователь может ввести запрос или загрузить свои заметки, и система сгенерирует практические тесты, учебные руководства и карточки. Однако все эти функции завязаны на облачную экосистему и платную подписку (от 2325 руб. в год). Более того, AI-сценарии Quizlet ориентированы преимущественно на английский и другие широко распространенные языки, а работа с русскоязычным контентом реализована на базовом уровне и не включает специализированных инструментов для OCR с поддержкой кириллицы. Глубокая кастомизация для конкретных учебных задач в Quizlet невозможны.

Duolingo, в свою очередь, фокусируется на языковом обучении и использует адаптивный алгоритм на основе Half-Life Regression (HLR). Платформа эффективно подбирает интервалы повторения для слов и грамматических конструкций, а также внедряет AI-персонализацию через движок Birdbrain. Однако Duolingo — это закрытая платформа, которая не предоставляет возможности для загрузки собственных произвольных учебных материалов (например, PDF-файлов с лекциями по истории или биологии). Пользователь ограничен готовыми курсами, предлагаемыми самой платформой.

Таким образом, ни одна из рассмотренных платформ не предоставляет единого конвейера: импорт русского текста из фотографии или PDF и последующее преобразование в колоду карточек. Разрабатываемое приложение SigmaCards является актуальным для русскоязычного рынка, так как с самого начала проектируется как целостная система, объединяющая OCR-компонент с поддержкой кириллицы, генеративный AI и современный алгоритм FSRS.

В следующем подразделе дается постановка задачи. Приложение SigmaCards позволяет создавать собственные колоды карточек, организовывать учебные сессии, отслеживать прогресс и получать статистику обучения. Ключевой особенностью является ML-сервис, автоматизирующий генерацию карточек из загружаемых материалов. В рамках данной работы выполняется разработка именно этого ML-сервиса.

Далее рассматриваются варианты подходов к решению задачи. ML-сервис решает две принципиально разные задачи: распознавание текста из изображений и PDF-файлов, а также автоматическая генерация вопросно-ответных пар из распознанного текста. Для OCR-компонента рассматриваются основные кандидаты.

Tesseract OCR — один из известных open-source OCR-движков, поддерживаемых Google. Обладает широкой языковой поддержкой, включая русский язык, хорошо задокументирован и легко встраивается в Python-проекты. Тем не менее Tesseract демонстрирует заметное снижение качества на сложных изображениях: при наличии шума, неравномерного освещения, искажений перспективы или рукописных элементов точность распознавания существенно падает. В сравнительных тестах на реальных фотографиях конспектов и сканах документов Tesseract нередко уступает современным решениям.

EasyOCR — относительно молодая библиотека на базе глубоких нейронных сетей, поддерживающая свыше 80 языков. Ее основное достоинство — простой API и встроенная поддержка GPU-ускорения. Однако по метрикам точности на русскоязычных документах EasyOCR уступает более специализированным решениям, а объем модели в памяти оказывается сравнительно велик, что сказывается на скорости инициализации контейнера.

Google Cloud Vision API и AWS Textract — облачные сервисы, предоставляющие OCR-функциональность через REST-интерфейс. Оба обеспечивают высокую точность, в том числе для кириллицы, и не требуют разворачивания

модели на собственном сервере. Однако их использование влечет регулярные финансовые расходы, зависимость от доступности стороннего API и потенциальные ограничения, связанные с передачей учебных данных пользователей на зарубежные серверы. Для прототипа с открытым исходным кодом и возможностью self-hosted-развертывания это существенный недостаток.

PaddleOCR — это open-source OCR-фреймворк, разработанный компанией Baidu в рамках платформы PaddlePaddle. В серии версий PP-OCRv2 и PP-OCRv3 авторы последовательно повышали метрику Hmean: по данным публикации, переход с PP-OCRv2 на PP-OCRv3 обеспечил прирост Hmean приблизительно на 5% при сопоставимой скорости обработки. PaddleOCR реализует полный конвейер из трех этапов: детектор областей текста (DB/DB++), классификатор ориентации строки и распознаватель (SVTR). Фреймворк поддерживает более 80 языков, включая русский, распространяется по лицензии Apache-2.0 и допускает запуск как на CPU, так и с GPU-ускорением в Docker-контейнере.

По совокупности критериев — открытая лицензия, верифицированная поддержка кириллицы и возможность полностью локального развертывания — для ML-сервиса SigmaCards выбран PaddleOCR.

После извлечения текста из документа необходимо сформировать из него вопросно-ответные пары. Эта задача решается с помощью большой языковой модели (LLM), которой передается фрагмент текста и системный промпт с инструкцией по формату вывода. Ниже рассмотрены основные варианты.

Локальные open-source модели (Mistral, LLaMA, Qwen) — семейства моделей с открытыми весами, пригодные для локального запуска через `ollama` или `vllm`. Локальный запуск требует значительных вычислительных ресурсов (GPU с объемом видеопамяти от 16 ГБ для моделей среднего размера), а качество генерации на русском языке у большинства западных open-source моделей заметно уступает специально дообученным на русскоязычных корпусах решениям.

YandexGPT — это отечественная языковая модель, доступная через API Yandex Cloud. Обеспечивает хорошее качество работы с русскоязычным текстом, однако, как и OpenAI GPT, предполагает облачное использование, передачу данных на сторонние серверы и тарификацию по токенам. Поддержка структурированного JSON-вывода реализована, однако документация API уступает по зрелости решениям с более широкой экосистемой.

Сбер GigaChat — большая языковая модель, разработанная Сбербанком.

Доступна через REST API с несколькими уровнями доступа: бесплатный тариф для разработки и коммерческие тарифы для продуктивного использования. GigaChat изначально обучена преимущественно на русскоязычных текстах, что обеспечивает высокое качество понимания и генерации текста на русском языке. Модель поддерживает режим структурированного ответа, допускает системный промпт и способна стабильно генерировать JSON-объекты заданного формата — что является ключевым требованием для генерации карточек.

Для SigmaCards выбрана модель GigaChat: ключевыми факторами стали высокое нативное качество работы с кириллицей, наличие бесплатного тарифа для прототипирования и поддержка структурированного вывода.

Второй раздел работы посвящен практической реализации ML-сервиса. Для разработки выбран язык Python 3.11, обладающий развитой экосистемой для создания ML/AI-приложений. Веб-часть реализована на FastAPI, что позволяет быстро создавать асинхронные REST API с автоматической Swagger-документацией и встроенной валидацией данных через Pydantic. Запуск приложения выполняется через ASGI-сервер Uvicorn. Для асинхронного взаимодействия с другими частями системы используется Kafka через библиотеку aiokafka. Redis используется для хранения состояния задач генерации. Проект контейнеризован с помощью Docker.

В части системных требований отмечается, что основным ресурсоемким местом сервиса является PaddleOCR, выполняющий OCR-распознавание локально. При использовании PaddlePaddle на CPU для минимальной работы рекомендуется не менее 2 vCPU и 8 ГБ оперативной памяти. Для более стабильной работы предпочтительны 4–8 vCPU и 16 ГБ RAM. При использовании GPU-ускорения требуется NVIDIA-карта с 6–8 ГБ VRAM, что существенно снижает время OCR-распознавания.

В подразделе архитектуры сервиса рассмотрена его общая структура. Сервис предоставляет HTTP API на FastAPI и поддерживает асинхронную обработку задач через Kafka. Архитектурно приложение состоит из нескольких модулей: API-слоя, сервисного слоя, слоя интеграции с GigaChat, модуля обработки PDF, Kafka-воркера и Redis-хранилища состояния. API-слой принимает запросы и выполняет первичную валидацию данных. Сервисный слой формирует prompt в зависимости от выбранного режима генерации. Затем запрос отправляется во внешний GigaChat API, а полученный ответ преобразуется в набор карточек.

Сервис поддерживает пять режимов генерации карточек: ключевые термины и определения (`key_terms`), факты, даты и числа (`facts`), карточки с пропусками (`fill_blank`), тестовые вопросы (`test_questions`) и объяснение концепций (`concepts`).

Функциональные требования к сервису включают: прием текстового учебного материала для генерации карточек; поддержку генерации в нескольких режимах; формирование prompt-а для языковой модели в зависимости от выбранного режима; обращение к GigaChat API для генерации содержимого карточек; обработку ответа языковой модели и преобразование его в структурированный список карточек; извлечение текста из PDF-документов с применением OCR-распознавания для страниц без текстового слоя; возврат результата в формате, совместимом с системой SigmaCards; обработку ошибок на всех этапах конвейера.

В подразделе описания основных Pydantic-моделей рассматривается механизм валидации данных. Модель `CardGenerationRequest` описывает входной запрос и содержит исходный текст для обработки, режим генерации и количество создаваемых карточек, причем для полей заданы ограничения на длину. Модель `Card` представляет учебную карточку с вопросом и ответом. Модель `CardGenerationResponse` содержит статус выполнения операции и список сформированных карточек. Модель `ErrorResponse` включает статус операции, код ошибки и текстовое описание проблемы.

В подразделе HTTP-взаимодействия описан FastAPI-слой сервиса. Входной точкой приложения является файл `app/main.py`, в котором создается экземпляр FastAPI-приложения. В сервисе используется механизм lifespan, позволяющий выполнять действия при запуске и остановке приложения, в том числе корректно запускать и останавливать Kafka-воркер и освобождать ресурсы. Настроен CORS middleware для возможности обращения frontend-клиентов к API из браузера. Корневой endpoint возвращает базовую информацию о сервисе: название, версию, окружение и состояние Kafka-интеграции. Endpoint `/health` используется для проверки работоспособности сервиса.

В подразделе сервисного слоя описаны ключевые внутренние компоненты. `PromptService` расположен в модуле `app/services/prompt_service.py` и отвечает за формирование запросов к языковой модели: он выбирает подходящий шаблон в зависимости от режима генерации и подставляет в него исходный

текст и количество карточек. Системный промпт объясняет языковой модели ее роль AI-ассистента для создания учебных карточек и задает ожидаемый формат ответа в виде JSON. Пользовательские промпты содержат исходный текст, требуемое количество карточек, описание типа карточек и дополнительные требования.

`LLMService` отвечает за интеграцию с GigaChat и содержит всю логику отправки запроса, получения ответа, парсинга JSON и проверки результата. При инициализации создается клиент GigaChat, параметры подключения берутся из конфигурации приложения. Метод `generate_cards` формирует объект запроса, содержащий системные инструкции и конкретный запрос на генерацию, отправляет его в GigaChat и извлекает содержимое ответа. После получения ответа каждая карточка проходит дополнительную проверку: контролируется наличие обязательных полей, тип структуры, минимальная и максимальная длина вопроса и ответа, соответствие количества карточек запрошенному значению.

`TextLoaderService` загружает текстовые материалы по URL, что особенно удобно для асинхронной обработки через Kafka: в сообщении передается ссылка на текст, а ML-сервис самостоятельно загружает содержимое перед генерацией карточек. Сервис выполняет асинхронный HTTP-запрос, следует редиректам, проверяет HTTP-статус ответа, удаляет лишние пробелы и контролирует максимальный размер текста — по умолчанию 10 000 символов.

`PDFLoaderService` предназначен для извлечения текста из PDF-документов. Сначала сервис пытается получить обычный текстовый слой страницы с помощью PyMuPDF. Если текстовый слой пустой, страница рендерится в изображение и передается на OCR-распознавание через PaddleOCR. Для работы с PaddleOCR используется паттерн Singleton с блокировкой потоков: OCR инициализируется единожды, а при последующих вызовах возвращается уже готовый экземпляр. При распознавании отключены дополнительные этапы обработки ориентации документа, а язык распознавания установлен как русский. Основная нагрузка сервиса связана именно с OCR, поскольку распознавание выполняется локально и требует значительных ресурсов CPU или GPU.

`KafkaWorker` реализует асинхронную интеграцию ML-сервиса с другими компонентами системы. При инициализации создаются внутренние сервисы — `TextLoaderService`, `PDFLoaderService`, `PromptService`, `LLMService` и

GenerationStateService, а также семафор, ограничивающий количество одновременно обрабатываемых сообщений (в текущей реализации — не более двух). При запуске воркер создает Kafka-consumer и producer: consumer подписывается на топики генерации карточек и извлечения текста из PDF, producer используется для отправки результатов в response-топики. После получения сообщения воркер определяет тип задачи по топику: для генерации карточек загружается текст по ссылке, формируется prompt и передается в LLMService; для извлечения текста из PDF вызывается PDFLoaderService. В конце результат отправляется в соответствующий response-топик. После успешной обработки выполняется commit Kafka-offset.

GenerationStateService контролирует состояние асинхронных задач через Redis. Для каждой задачи формируется ключ вида generation:skip:{generation_id}. Если по этому ключу в Redis хранится значение «1», задача считается помеченной на пропуск, и KafkaWorker не выполняет дальнейшие действия: загрузку текста, OCR-распознавание или обращение к GigaChat API. Такой механизм особенно важен в асинхронной архитектуре, где задача может оказаться неактуальной к моменту обработки, например если пользователь отменил генерацию.

В подразделе развертывания сервиса описывается его контейнеризация с помощью Docker. В качестве базового образа используется python:3.11-slim. В контейнер копируется файл зависимостей, устанавливаются необходимые системные библиотеки и Python-пакеты, после чего копируется исходный код приложения. Запуск сервиса внутри контейнера выполняется через Uvicorn. В docker-compose.yml описан сервис sigmacards-ml, который пробрасывает порт 8000, монтирует исходный код и автоматически перезапускается при сбоях. После запуска API доступно по адресу http://localhost:8000, а Swagger-документация — по адресу http://localhost:8000/docs.
Если Kafka-интеграция не требуется, ее можно отключить параметром KAFKA_ENABLED=false, и сервис будет работать только через HTTP API.

В подразделе документации по интеграции описаны два способа взаимодействия с ML-сервисом. HTTP API используется для синхронных сценариев:

— GET / — получение базовой информации о сервисе;
— POST /api/v1/cards/generation — основной endpoint для генерации карточек, принимающий исходный текст, режим генерации и количество

карточек;

- `GET /api/v1/cards/modes` — получение списка доступных режимов генерации;
- `GET /api/v1/cards/modes/{mode_name}` — получение описания конкретного режима вместе с примером входного текста и примером карточки.

Для асинхронных сценариев используются четыре Kafka-топика: `ml.cards.generate.request` и `ml.cards.generate.response` для генерации карточек, `ml.cards.extract.request` и `ml.cards.extract.response` для извлечения текста из PDF. Сообщение запроса на генерацию содержит идентификатор запроса, уникальный идентификатор задачи, ссылку на текстовый файл, режим генерации и количество карточек. В ответном сообщении возвращается массив карточек, каждая из которых содержит лицевую сторону с вопросом и обратную с ответом.

В заключение следует отметить, что в рамках данной работы был разработан и интегрирован ML-сервис для мобильного приложения SigmaCards, обеспечивающий автоматическую генерацию учебных карточек на основе больших языковых моделей. Все поставленные задачи были выполнены:

- проведен анализ методов автоматической генерации на основе LLM и выбрана модель GigaChat;
- выполнен анализ OCR-решений с поддержкой кириллицы и выбран PaddleOCR;
- разработана архитектура ML-сервиса как независимого масштабируемого компонента на базе FastAPI с асинхронным взаимодействием через Apache Kafka и хранением состояния в Redis;
- реализован конвейер обработки входных материалов с поддержкой пяти режимов генерации;
- сервис контейнеризирован с помощью Docker, для него подготовлена документация по интеграции через HTTP API и Kafka.

Результатом работы является полноценный ML-сервис, снижающий порог входа в технологии интервального повторения за счет автоматизации создания учебных карточек из неструктурированных материалов: текстов, PDF-файлов и изображений с кириллическим содержимым. Данная работа была представлена на программе «Стартап как диплом» и получила экспертную оценку «отлично».